Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Отчёт по лабораторной работе № 2

Дисциплина: Проектирование мобильных приложений

Тема: Activity Lifecycle. Alternative resources.

Выполнил студент гр. 3530901/90201 _____ Е.К. Борисов
(подпись)

Принял старший преподаватель _____ А.Н. Кузнецов
(подпись)

“_____” _____ 2021 г.

Санкт-Петербург
2021

Оглавление

Цели:

- Познакомиться с жизненным циклом Activity
- Изучить основные возможности и свойства alternative resources

Введение

Activity

Acitvity – это главный компонент Android-приложения. *Activity* представляет собой окно с пользовательским интерфейсом. В основном окно Activity полностью заполняет экран, однако оно может находиться поверх других окон или быть меньше экрана. Обычно одна активность представляет собой один экран в приложении.

Приложение чаще всего состоит из нескольких экранов и соответственно содержит несколько acivity. Одна из активностей приложения бывает главной и появляется при запуске приложения. Каждая из активностей может запускать другую активность.

Activity Lifecycle

Каждая активность имеет свой жизненный цикл, может находиться в одном из нескольких состояний, которые меняются под влиянием различных факторов, например действий пользователя или того, что происходит в самом приложении.

Класс Activity предоставляет ряд коллбэков, которые сообщают активности о том, что состояние изменилось. Каждый коллбэк позволяет выполнять определённую действия, соответствующие заданному состоянию, таким образом, в них можно

описать как должна вести себя активность на конкретных этапах жизненного цикла.

Класс Activity предоставляет базовый набор из шести коллбэков: onCreate(), onStart(), onResume(), onPause(), onStop() и onDestroy(). Система вызывает каждый из этих коллбэков как только активность переходит в новое состояние. На рисунке 1, представлена схема перехода из различных состояний.

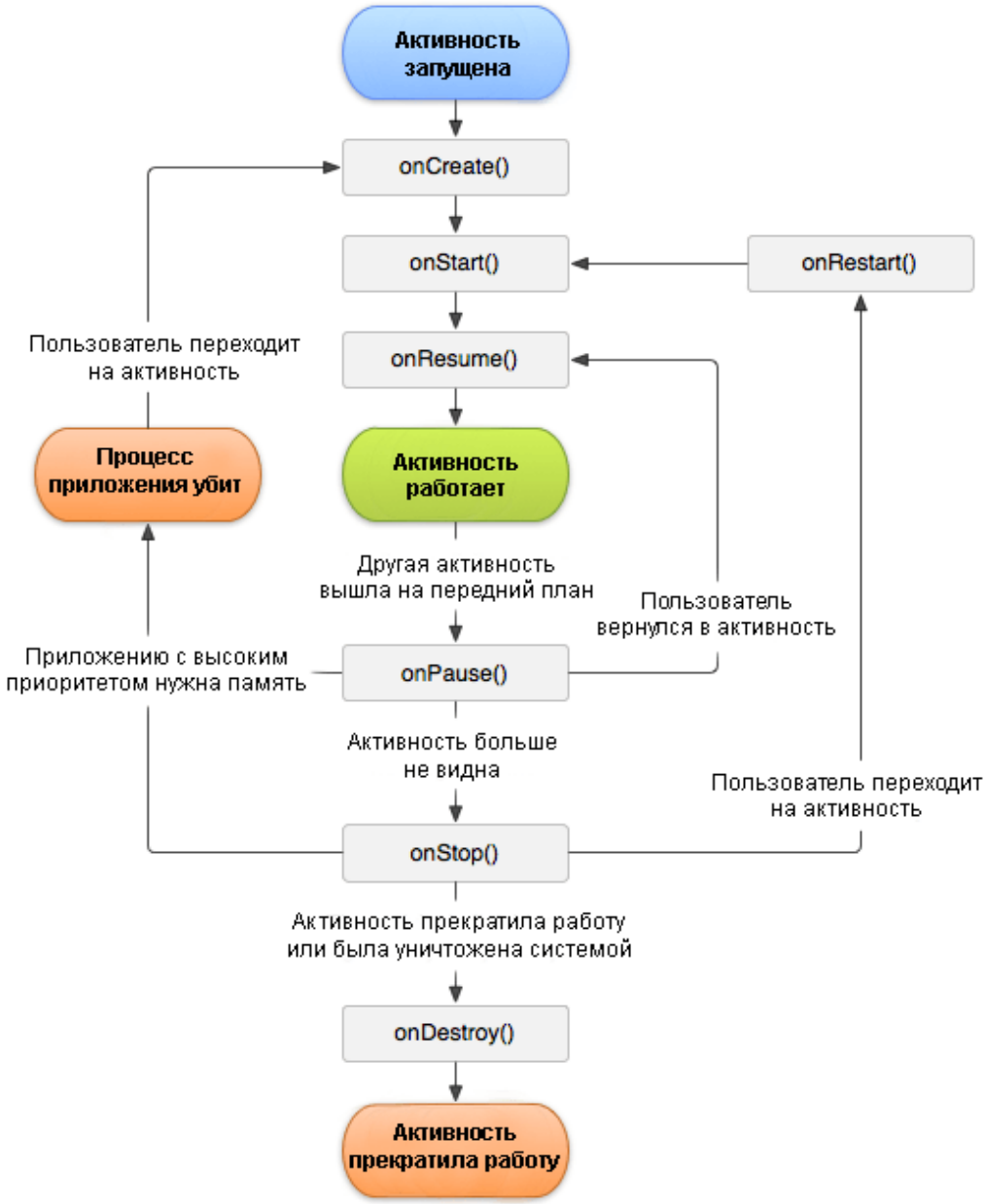

Рис. 1 Жизненный цикл Activity

В зависимости от сложности активности, не всегда обязательно реализовывать все методы жизненного цикла. Однако важно понимать каждый из них и реализовывать те, которые обеспечивают правильную работу приложения.

onCreate()

Данный метод срабатывает при создании активности и он является обязательным, так как здесь происходит первоначальная настройка активности. В данном методе реализуются действия, которые должны происходить при запуске приложения, которые выполняются один раз, например, привязка данных к спискам. Этот метод принимает в качестве параметра savedInstanceState, который представляет собой объект Bundle, содержащий ранее сохраненное состояние активности. Если активность ранее не существовала, значение объекта Bunde будет равно null.

После того, как onCreate() завершит выполнение, активность переходит в состояние Started и система следом вызывает onStart() и onResume().

onStart()

Когда активность переходит в состояние Started, система вызывает этот метод. Вызов onStart() делает активность видимой для пользователя, так как приложение готовится к переходу активности на передний план и становится интерактивной. Например, здесь можно реализовывать код, который будет поддерживать пользовательский интерфейс.

Метод onStart() завершается очень быстро и, как и с onCreate(), активность не остаётся в состоянии Started, а переходит в состояние Resumed, после чего система вызывает метод onResume().

onResume()

Когда активность переходит в состояние Resumed, она выходит на передний план, а затем система вызывает метод onResume(). Это состояние, в котором приложение взаимодействует с пользователем. Приложение остается в этом состоянии, пока не произойдёт что-то, что переключит фокус с приложения. К таким событиям можно отнести, например, входящий вызов или выключение экрана устройства.

Когда происходит событие, прерывающее текущее состояние, активность переходит в состояние Paused и система вызывает метод onPause().

onPause()

Данный метод сигнализирует о том, что пользователь покидает текущую активность. Существует несколько причин, по которым активность может войти в это состояние:

- Происходят события, которые переключают фокус с приложения.
- Начиная с **Android 7.0 (API 24)** приложения можно запускать в многооконном режиме и в таком случае система сконцентрирована на одном из приложений, и работа всех остальных приложений приостанавливается
- Открывается новая полупрозрачная активность. Активность еще остается видимой, но система не сфокусирована на нем.

Данный метод можно использовать для освобождения системных ресурсов, однако, так как приложение может быть частично видимым, этого делать не стоит, помимо этого, выполнение onPause() происходит очень быстро, поэтому не стоит реализовывать в нем операции, требующие много времени, так как они могут не успеть завершиться для этих действий лучше использовать **onStop()**.

onStop()

Когда активность больше не видна пользователю, она переходит в состояние Stopped, и система вызывает метод onStop(). Это может произойти, например, когда вновь запущенная активность охватывает весь экран. Система также может

вызвать onStop(), когда активность завершила свою работу и вот-вот будет уничтожена.

Как говорилось ранее, в этом методе должно происходить освобождение ресурсов и их регулировка. Его также можно использовать для относительно затратных в плане расхода CPU операций, например, для сохранения информации в бд

В состоянии Stopped активность либо возвращается для взаимодействия с пользователем, либо полностью завершается. Если активность возвращается, система вызывает onRestart(). Если активность завершается, система вызывает метод onDestroy(), при этом она также сохраняет состояния объектов View в Bundle.

onDestroy()

Метод onDestroy() вызывается до того, как активность будет уничтожена. Система вызывает этот метод по следующим причинам:

Активность завершает свою работу поскольку пользователь закрывает активность либо в приложении вызывается метод finish().
Система временно уничтожает активность из-за изменения конфигурации (например, поворот устройства или использование многооконного режима).
Если активность прекращает работу, то onDestroy() — это последний коллбэк жизненного цикла активности. Если onDestroy() вызывается в результате изменения конфигурации, система немедленно создаёт новый экземпляр активности и затем вызывает onCreate() в новом экземпляре.

Метод onDestroy() освобождает все ресурсы, которые ещё не были освобождены в методах ранее, такими как onStop().

Также есть дополнительный вызов onRestart() – вызывается после onStop(), когда текущий Activity должен быть снова отображен пользователю. После него следуют onStart() и onResume(). В продвинутых приложениях является хорошим знаком, что этот метод не вызывается.

Задача 1

Цель задания: продемонстрировать жизненный цикл Activity на любом нетривиальном примере.

Мы создали андройд приложение с двумя активностями и добавили кнопку перехода между этими двумя компонентами также мы добавили в методы обратных вызовов логирование. В onCreate() были добавлены также обработчики событий для кнопок. Для отслеживания вызовов мы используем инструмент LogCat. Запустим приложение и перейдем из главного окна в другое и обратно и посмотрим на результат.

Листинг 1: Запуск приложения.

```
2021-10-17 03:54:00.997 6026-6026/com.example.lab_3 D/States: MainActivity: onCreate()
2021-10-17 03:54:00.998 6026-6026/com.example.lab_3 D/States: MainActivity: onStart()
2021-10-17 03:54:00.999 6026-6026/com.example.lab_3 D/States: MainActivity: onResume()
```

Мы видим, что при запуске приложения у нас запускается главное активити, проходя через методы onCreate(), onStart(), onResume() как и ожидалось в соответствии с рисунком 1.

Теперь перейдем во второе активити и обратно.

Листинг 1: Переход в другое активити.

```
Переход в SecondActivity
2021-10-17 03:59:19.340 6026-6026/com.example.lab_3 D/States: MainActivity: onPause()
2021-10-17 03:59:19.369 6026-6026/com.example.lab_3 D/States: SecondActivity: onCreate()
2021-10-17 03:59:19.373 6026-6026/com.example.lab_3 D/States: SecondActivity: onStart()
2021-10-17 03:59:19.373 6026-6026/com.example.lab_3 D/States: SecondActivity: onResume()
2021-10-17 03:59:19.841 6026-6026/com.example.lab_3 D/States: MainActivity: onStop()
Переход обратно в MainActivity
2021-10-17 04:03:21.529 6026-6026/com.example.lab_3 D/States: SecondActivity: onPause()
2021-10-17 04:03:21.557 6026-6026/com.example.lab_3 D/States: MainActivity: onCreate()
2021-10-17 04:03:21.558 6026-6026/com.example.lab_3 D/States: MainActivity: onStart()
2021-10-17 04:03:21.559 6026-6026/com.example.lab_3 D/States: MainActivity: onResume()
2021-10-17 04:03:22.054 6026-6026/com.example.lab_3 D/States: SecondActivity: onStop()
```

Как мы видим при переходе в другое activity сначала вызывается метод onPause(), затем происходит создание и отображение второго активити onCreate() -> onStart() -> onResume() и только после этого срабатывает метод toStop() у первого активити, когда оно становится полностью невидимым для пользователя. То же самое происходит при переходе из второго активити в первое.

Также мы использовали виджет EditText у первого layout. Мы ввели текст в тектовое поле и при переходе из первого активити во второе и обратно, значение в EditText было утеряно. При этом, если ввести текст, закрыть и заного открыть приложение, значение в EditText сохраняется и происходят следующие переходы из состояний.

Листинг 1: Закрытие и открытие приложения.

```
2021-10-17 05:52:46.231 1921-1921/com.example.lab_3 D/States: MainActivity: onPause()
2021-10-17 05:52:46.971 1921-1921/com.example.lab_3 D/States: MainActivity: onStop()
2021-10-17 05:52:50.731 1921-1921/com.example.lab_3 D/States: MainActivity: onRestart()
2021-10-17 05:52:50.736 1921-1921/com.example.lab_3 D/States: MainActivity: onStart()
2021-10-17 05:52:50.737 1921-1921/com.example.lab_3 D/States: MainActivity: onResume()
```

Это также соответствует схеме представленной на рисунке 1.

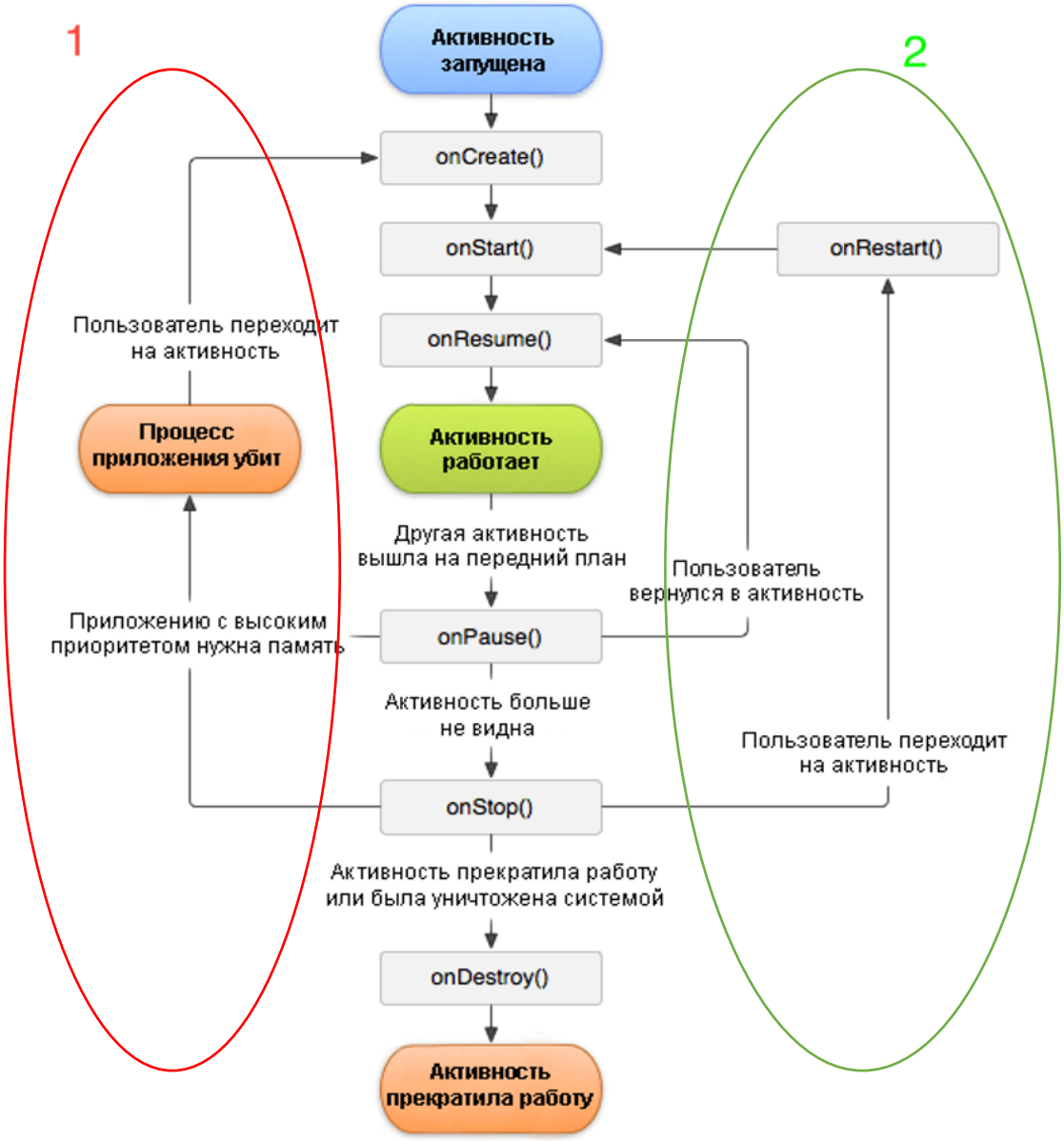

Рис 2. Возвращение к предыдущему активити.

В случае перехода в другое активити происходит первая цепочка событий, в случае же простого закрытия и открытия приложения - вторая.

Задача 2

Использование альтернативного ресурса screen size.

Допустим у нас есть приложение на в котором пользователи читают информацию о фильмах, в нем у нас есть логотип нашего приложения, под ним постер фильма и его название, и под постером надпись описание и под ней описание самого фильма, мы описали такой макет и на стандартном размере экрана с разрешением 1080x1920 и диагонально 5.0 и плотностью 420 dpi он будет как мы и задумывали, но например если мы запустим это же приложение на телефоне с меньшим экраном на разрешении 240x320 и диагонали 2.7 наш шаблон будет выглядеть абсолютно не так, как мы хотим.

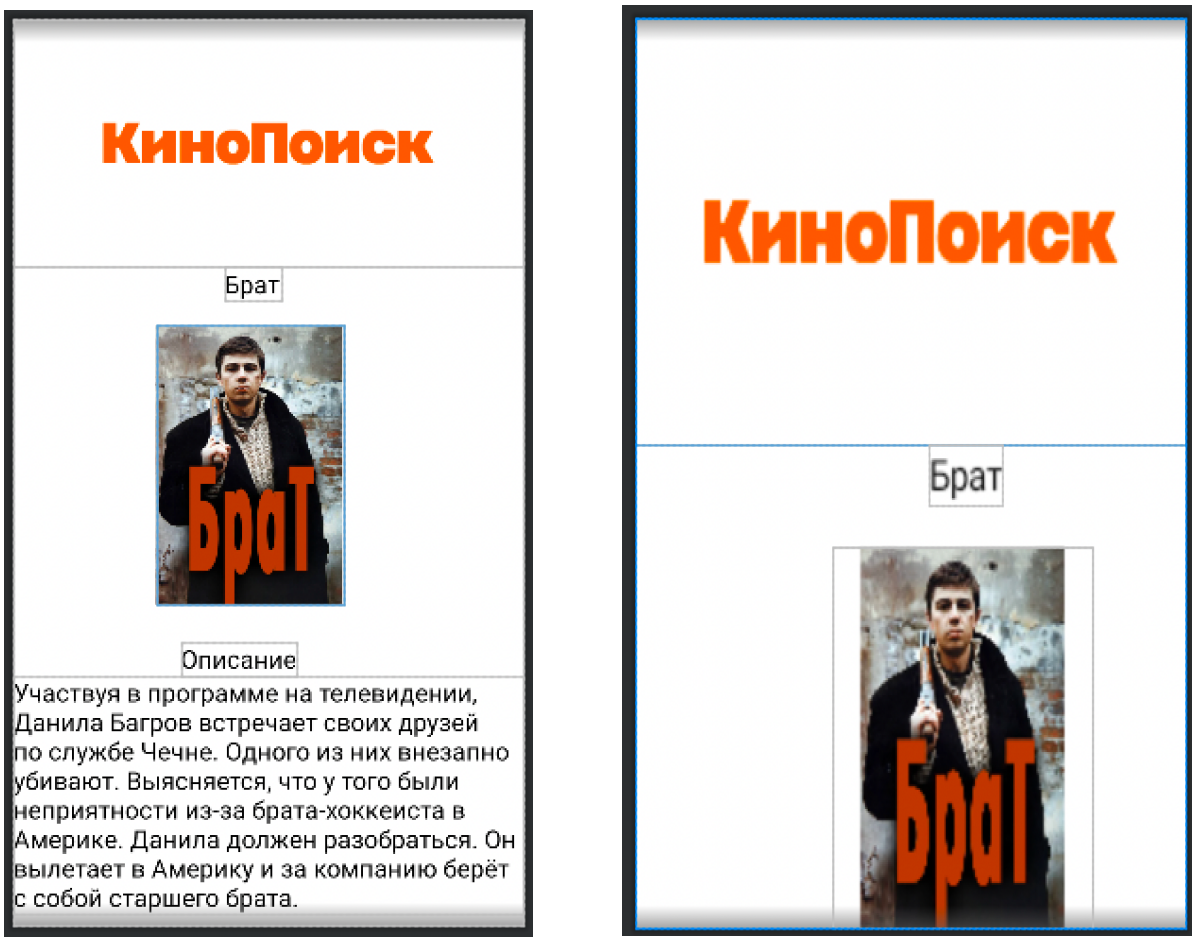

Рис. 3 Пример приложения для различных размеров экрана.

С помощью данного ресурса мы можем описать шаблоны для различных размерностей экрана а именно:

- small: Экраны, похожие на экран QVGA низкой плотности. Минимальный размер макета для небольшого экрана составляет примерно 320x426 dp.
- normal: Экраны, похожие на экран HVGA средней плотности. Минимальный размер макета для обычного экрана составляет примерно 320x470 dp.
- large: Экраны, похожие на экран VGA средней плотности. Минимальный размер макета для большого экрана составляет примерно 480x640 dp.
- xlarge: Экраны, которые значительно больше, чем традиционный экран HVGA средней плотности. Минимальный размер макета для большого экрана составляет примерно 720x960 dp.

Таким образом, в зависимости от того на каком устройстве запущено наше приложение, будет выбираться наиболее подходящий шаблон и отображение элементов шаблона будет корректным.

Задача 3.

Выбрать ресурс по алгоритму для best-matching resource.

==========

Вариант 5:

==========

Конфигурация устройства	Конфигурация ресурсов:
LOCALE_LANG: en LOCALE_REGION: rCA SCREEN_SIZE: small SCREEN_ASPECT: long ROUND_SCREEN: notround ORIENTATION: port UI_MODE: desk NIGHT_MODE: notnight PIXEL_DENSITY: xxhdpi TOUCH: notouch PRIMARY_INPUT: qwerty NAV_KEYS: trackball PLATFORM_VER: v25	(default) normal-notouch-12key en-rUS-notlong-port-notouch-nonav-v27 en-appliance-night notround-port round-tvdpi-trackball en-large-notround-notouch-v25 rFR-long-round en-xlarge-notlong-notround-vrheadset-mdpi small-notlong-round rFR-television-mdpi-nonav

Первым делом вычеркивеим ресурсы противоречащие конфигурации устройства:

(default)

~~normal-notouch-12key~~ – противоречит размеру экрана

~~en-rUS-notlong-port-notouch-nonav-v27~~ – противоречит региону

~~en-appliance-night~~ – противоречит UI моду

notround-port

~~round-tvdpi-trackball~~ – противоречит формату экрана

~~en-large-notround-notouch-v25~~ – противоречит размеру экрана

~~rFR-long-round~~ – противоречит региону

~~en-xlarge-notlong-notround-vrheadset-mdpi~~ – противоречит размеру экрана

~~small-notlong-round~~ – противоречит аспекту экрана

~~rFR-television-mdpi-nonav~~ - противоречит региону

notround-port – ресурс, который не противоречит конфигурации, вычеркиваем все, где нет notround. ~~(default)~~ . Следовательно Best-matching resource – notround-port.

Задача 4.1

Нужно провести ручное тестирование приложения continuewatch. Тестирование было проведено на эмуляторе и выявлены следующие ошибки:

- Не следует использовать конкатенацию с setText() вместо этого следует использовать плейсхолдеры в строковых ресурсах
- При запуске приложения мы вначале видим TextView “Hello World”
- Приложение считает секунды, когда оно не отображается на экране.
- При смене ориентации счетчик сбрасывается
- При смене ориентации мы также вначале видим TextView “Hello World”

Задача 4.2

Исправим найденные ошибки, пойдем по порядку, используем ресурс plurals с помощью которого можно вывести строку в зависимости от входного параметра, чтобы исправить ошибку с setText.

```
<plurals name="secCounter">
    <item quantity="other">Seconds elapsed: %d</item>
</plurals>
```

Рис. 4 Использование plurals ресурса.

%d – означает, что мы будем передавать десятичное целое число в качестве параметра.

В Activity для получения необходимой строки используем метод getQuantityString(int resId, int quantity, Object… formatArgs), принимающий в качестве параметров ссылку на сам ресурс, а также дважды наш счетчик, если бы строка не использовала форматирование, то передача третьего параметра была бы не нужной.
Также удалим атрибут android:text из xml файла, чтобы эта надпись не появлялась вначале работы приложения.

Теперь сделаем так, чтобы приложение не считало секунды, когда оно не отображается на экране. Так как, когда мы сворачиваем приложение, система вызывает onPause, а когда мы возвращаемся в приложение onResume, мы можем переопределить эти два метода, добавив в них флаг, который будет сигнализировать о том, что приложение закрыто, или перекрыто другим приложением, поэтому нужно остановить счет.

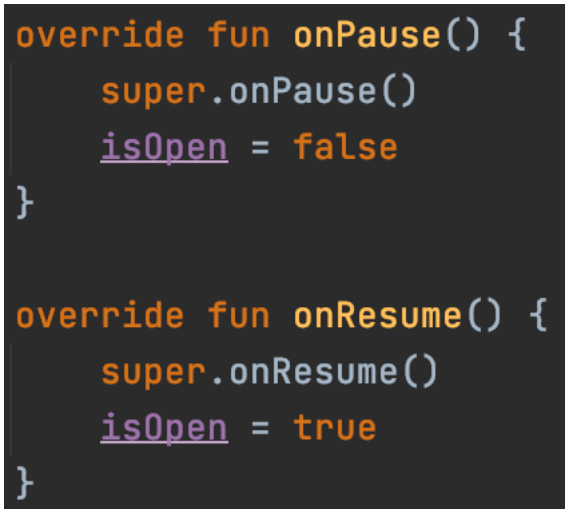

Рис 5. Исправление счета при выходе из приложения.

Последняя ошибка возникает по той же причине, почему значение в EditText в задании один не сохраняло свое значение при переходе на новое активити, старое активити уничтожается и создается новое активити, теряя данные.

Чтобы исправить эту ошибку, можно воспользоваться объектом savedInstanceState, который приходит в качестве параметра в функцию onCreate и имеет тип Bundle, который представляет собой набор пар ключ (String) – значение (Parcelable) и может быть использован для сохранения предыдущего состояния activity.

Для сохранения данных в объект savedInstanceState используется функция onSaveInstanceState, переопределим эту функцию, вызвав в теле функции run объекта Bundle вызовем функцию putInt для создания элемента коллекции с ключем “KEY” и значением счетчика secondElapsed в качестве значения. Также переопределим функцию onRestoreInstanceState, которая вызывается при старте активити, в том случае, если имеется Bundle, сохраненный функцией onSaveInstanceState. В теле onRestoreInstanceState мы можем получить из бандла и присвоить нашему счетчику ранее сохраненное его значение.

```
override fun onSaveInstanceState(outState: Bundle) {
    outState.run {  this: Bundle
        putInt("KEY", secondsElapsed)
    }
    super.onSaveInstanceState(outState)
}

override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    secondsElapsed = savedInstanceState.getInt( key: "KEY")
    super.onRestoreInstanceState(savedInstanceState)
}
```

Рис. 6 Сохранение состояния счетчика при уничтожении activity.

Задача 4.3

Теперь используем другой способ сохранения данных – SharedPreferences.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main2)
    textSecondsElapsed = findViewById(R.id.textSecondsElapsed)
    backgroundThread.start()
    preferences = getSharedPreferences( name: "PREF", Context.MODE_PRIVATE)
    val button: Button = findViewById(R.id.button2)
    button.setOnClickListener {  it: View!
        val intent = Intent( packageContext: this, MainActivity::class.java)
        startActivity(intent)
    }
}

override fun onPause() {
    super.onPause()
    isOpen = false
    preferences.edit()
        .putInt("PREF",secondsElapsed)
        .apply()
}

override fun onResume() {
    super.onResume()
    isOpen = true
    secondsElapsed = preferences.getInt("PREF", 0)
}

}
```

Рис 6. Исправление приложения с SharedPreferences

Вначале приложения создаём новый файл с помощью метода getSharedPreferences (String name, int mode). В этом файле также содержаться пары ключ-значение. При этом если другое активити вызовет эту функцию с таким же именем будет возвращен тот же самый файл. Далее с помощью edit() в onPause() мы записываем в файл количество секунд и затем в методе onResume() с помощью getInt() получаем это значение. При этом при переходе между activity и даже завершении программы счетчик оставался с тем же значением, которое было до выключения.

Также в обеих activity добавили кнопку перехода между ними для удобного перехода.

Задача 4.4

В приложении 4.2 мы используем объект Bundle, который может содержать состояния активити до его разрушения и создания нового активити. Для этого нужно сохранить в него значение и затем в новом активити восстановить значение.

В приложении 4.3 мы используем SharedPreferences, создаем файл, который представляет собой файл с ассоциативным массивом, в связи с этим мы сохраняем в нем значение и даже при выключении приложения, мы получаем обратно то же самое значение, что было при выходе.

Вывод:

В данной лабораторной работе был изучен жизненный цикл Activity. Мы посмотрели как вызываются его коллбэки при различных действиях в системе и переопределили их для своих задач. Также были изучены альтернативные ресурсы, что это такое, для чего нужны и где следует применять. Узнали о методах onSaveInstanceState и onRestoreInstanceState , которые используются для сохранения состояния активити, а также поработали с SharedPreferences, который используется для этих же целей.